

Real-Time Clock

For the 89S8252 Flash Micro Board

Design by M. de Martelaer

This article was an entry to the Flash Competition of Elektor Electronics, and describes how a real-time clock (RTC) can be implemented in software, without using any dedicated RTC peripheral ICs. Since we also have to take account of drift, summertime and wintertime, an automatic synchronisation has been included with the DCF time signal transmitted from Frankfurt, Germany (range 1,000 miles).

All of the operations of this 'circuit' happen at the interrupt level. In this way it can be added easily to a larger application where it provides the correct time without disturbing the main program. Since a DCF receiver is used to obtain the time, there is no need for any buttons to set the time when the micro-processor system is started.

An application has been included as an example. Every second it:

- updates the LCD display with the correct time,
- transmits the time through the serial port.

To polish this project off, a Windows program was written (in Visual Basic), which displays this time on the PC screen and sets the PC clock.

The microcontroller software is written in Basic, which is converted into machine code by the BASCOM compiler. The following description should make it possible for anybody to convert the routines into another language of their choice.

Resources

The program requires just a single 16-bit timer for the clock and one I/O pin (digital input) to read the output of the DCF receiver.

The DCF seconds signal is provided by a ready-made receiver module from Conrad Electronics (www.int.conradcom.de), this

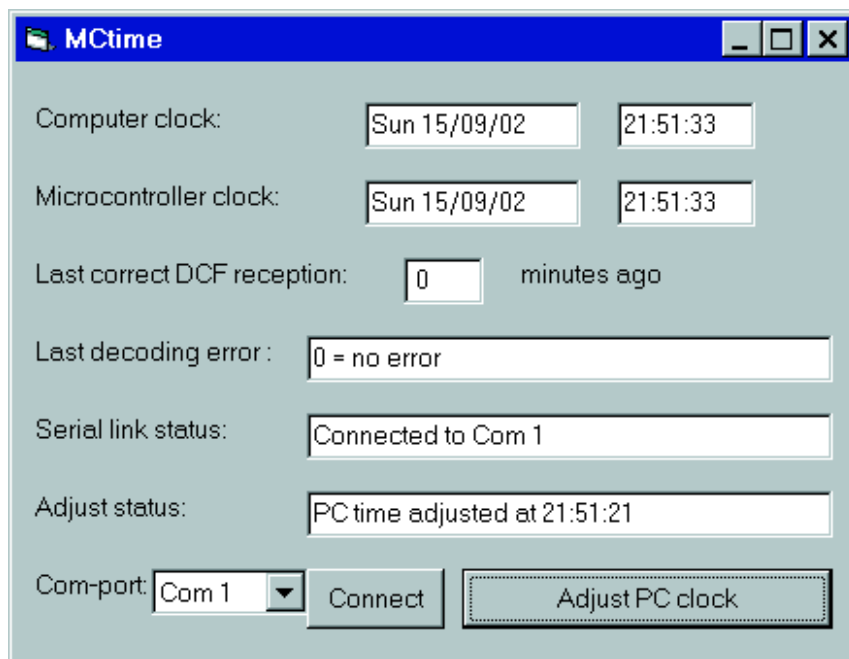


Figure 1. The PC clock can be synchronised using the 'Adjust PC Clock' button.

is cheaper (approx. £6.50) and easier to obtain than the individual parts and also prevents many problems.

The example program also makes use of several other I/O pins: an LED that flashes every second, an LED that shows the DCF pulses and finally the serial port to the PC.

Operation of the RTC

We've used an internal timer in interval mode. In this mode the timer/counter is incremented by 1 for every processor cycle, which is every 12 crystal pulses.

As we're using an 11.0592 MHz crystal this happens 11,059,200 / 12

= 921,600 times per second.

Since we're working with 16 bits, there will be an overflow after the timer reaches a value of 65,535. At that moment it will be reset to zero (or another predefined value) and an interrupt occurs. This interrupt should therefore occur $921,600 / 65,535 = 14.06$ times per second.

This, however, is not a good value to use for a clock tick. The nearest, higher, round value is 20 times per second, which gives a period of 50 milliseconds. This can be realised by configuring the timer such that it resets earlier: $921,600 / 20 = 46,080$.

If we can set up the timer such that it resets after 46,080 cycles, we obtain a clock that ticks every 50 milliseconds. There will still be some drift due to the inaccuracy of the crystal and the software overhead.

The value at which the timer resets is not adjustable: this is always 65,535. It is however possible to vary the starting value. This will therefore no longer be zero, but: $65,535 - 46,080 = 19,455$.

For a very precise clock, the starting value of the timer should be adjusted slightly to overcome its deficiencies. (See later). Increasing the starting value makes the timer reset earlier, causing the RTC to tick faster and vice versa. On top of this is the DCF synchronisation, which provides the correct time every minute when there is a good reception.

In the interrupt routine that is called every 50 milliseconds, it is a simple task to count the clock pulses to make seconds, the seconds to make minutes, and the minutes to make hours. Updating the date is much more difficult. For this reason the date is always set to what was sent by the DCF transmitter. If the DCF signal drops out, the date will remain the same. For most applications this is acceptable, but if a date routine is required you will have to add this yourself.

The DCF signal

The DCF77 transmitter near Frankfurt in Germany transmits an AM modulated signal at 77.5 kHz in the VLF band. The signal can be received at least a few times every day in most of continental Europe,

Scandinavia and the south-east of England. Every second a new bit is transmitted by reducing the amplitude of the carrier to 25 percent. The length of this amplitude reduction determines the value of the transmitted bit: 100 milliseconds is a '0' and 200 milliseconds is a '1'.

The DCF receiver converts the amplitude changes into pulses on its open collector outputs. There is a non-inverting output (high during the pulsing of the DCF transmitter) and an inverting output (drops to 0 Volt during the pulsing).

A description of the bits transmitted in the DCF77 signal can be found in the 'DCF-controlled LED clock' article, published in *Elektor Electronics* May 1998.

Decoding the DCF signal

The non-inverting output of the DCF receiver is connected to a digital input (in this case P3.3). We have already mentioned in the main description of the RTC that an interrupt is called every 50 ms. This routine is also used to poll the level at the input pin.

A '0' bit is recognised when the input is high for one or two clock ticks (100 ms pulse), and a '1' bit when it is high for three to five clock ticks. If we can't detect a high state for a period of 30 ticks (= 1.5 s), we know that the time message is at an end and that the next pulse signifies a change in minutes.

Of course only those bits that are required to extract the time and date are decoded from the 59 bits.

Implementation of the RTC

This takes place in the very first and last sections of the 'Clocktimeroverflow' routine. First the timer register has to be set. After an overflow this should be set to the calculated value. Since other interrupts may occur in the application which our clock is part of, it is best to add the reset value on to the value found in the timer register. The jump to our timer routine could have been delayed if another interrupt was already in progress (at the same or a higher priority). In the meantime the counter will have passed zero. An accurate

clock will take this into account: we read the value of the timer register, add on the reset value and write back the total.

The constant 'Reset_value' should in theory be set to 19,455 as was determined earlier. This is however increased by an offset value. In our example this is 150 and its function is to absorb the software overhead. After all, a number of instruction cycles pass between the reading and writing of the timer register, during which the timer doesn't run. The extreme case when the timer interrupt is serviced too late is checked for as well. The error variable 'Dcferror' then contains 15. In this case the RTC is not compatible with the main program: the interrupt routines of the main application are then taking more than 50 milliseconds to complete.

Since an 8052 processor has at least two interrupt priority levels, it is best to set the Timer0 interrupt to the highest priority. The other interrupts are set to the lower (default) priority. In this way a long interrupt in the main program won't affect the operation of the RTC. Instead, they are momentarily interrupted by the Clocktimeroverflow interrupt routine.

Operation of the DCF decoder

Many errors can occur during the minute-long coded message. These are made known through the error variable Dcferror; it is defined at the beginning of the BASIC file.

The BCD coded values are converted into decimal, pulse by pulse, in the 'Increment_dcf' routine. This is found right at the end.

You should also take care with the use of the Return statement in the interrupt routine: this may only occur once. The first Return statement is converted by BASCOM into the RETI assembler instruction. Any others following won't be, causing the program to fail, even though it compiles. If we want to exit the Clocktimeroverflow routine early we have to use a Goto instruction to jump to the label 'Clocktimeroverflow_return', which is where the only Return instruction is.

Reading the time

The time variables should be copied all at once to your own variables if you want to obtain a consistently reliable time. During this copying the timer interrupt should usually be disabled. In the example program (LCD and serial port) this isn't necessary because we only make use of the time directly after a change in seconds.

You should also make sure that the variable 'Dcfsince' is smaller than 255, otherwise

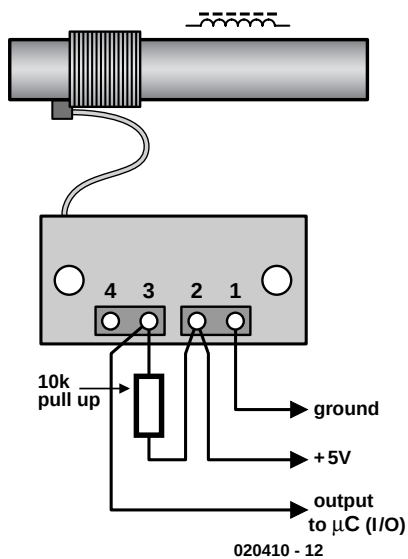


Figure 2. The DCF correction is provided by a ready-made receiver module.

the time is no longer valid.

The LCD and serial port

For diagnostic purposes the values of the RTC are shown on the LCD display. This LCD is connected for use with BASCOM (unfortunately, at the time of writing the author did not have the latest version of BASCOM which can configure the LCD pins on the Elektor board).

The first line of the display shows the time and the number of minutes since the last correction (S = since). The second line contains the date and the last error code (E = error):

```
14h1m10s S:0
Fr 18/10/2 E: 0
```

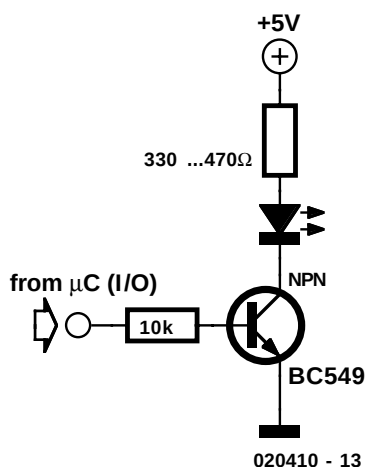


Figure 3. The LEDs are driven in the simplest way possible.

This is where we find a limitation of BASCOM: the length of the lines varies depending on the values of the time and date. The Print and LCD statements can't be used to display values in leading zero or leading space formats. The 'Print Using...' instruction found in Qbasic is not available in BASCOM either.

Every minute the date and time are transmitted from the serial port in the following format (ASCII):

```
Byte no.
1-3:   Header Text: 'DCF'
4:     Day of Week.
      1 = Monday, etc.
5-6:   Day (1-31)
7-8:   Month (1-12)
9-10:  Year (0-99)
11-12: Hours (0-24)
13-14: Minutes (0-60)
15-16: Seconds (0-60)
17-18: Error code from DCF
reception
19-21: Number of minutes
since the last correctly decoded
message
22-23: Checksum (0-98)
24-25: Terminator string:
      Carriage-Return /LineFeed
```

The checksum has been added because a parity bit cannot be implemented easily on the 8051. This value is calculated as follows: we first calculate the sum of all values sent (Day of Week + Day + Month + Year, etc.). We then take the remainder of this sum after dividing it by 99. This results in a number having two digits.

The 'Print Using...' statement can't be used here either. Any leading zeros are added manually because the message length has to be constant.

The PC program

The instructions for using the PC program are very simple: first choose a COM port and then press the 'Connect' button. If the last correctly received DCF message occurred less than 255 minutes ago, the PC clock can be synchronised using the 'Adjust PC clock' button (see Figure 1).

It is also possible for the connection and the synchronisation of the PC clock to happen automatically.

For this, you'll have to add the following parameters when starting the program from a DOS window, batch file or shortcut:

/comx: Automatic connection to the specified COM port, without the need to press the 'Connect' button. For example: /com1.

/boot: Automatic setting of the PC clock, without the need to press the 'Adjust PC clock' button; the program then exits. This option can only be used in conjunction with the COM port option. It is used to set the PC clock to the DCF clock every time Windows starts.

/min: The program starts 'minimized' and the window is not visible. This keeps the program in the background when used with the /boot option.

The baud rate of the COM port is automatically set to 19,200 bps, 8 bits, no parity, 1 stop bit.

Hardware connections

DCF input: P3.3 (req.)
Seconds LED: P3.4 (opt.)
DCF control LED: P3.5 (opt.)

A 16*2 LCD display connected for use with BASCOM (optional).

Figures 2 and 3 show the connections to the DCF receiver and the LEDs respectively.

Practical matters

Keep in mind that the PC, and especially the monitor, can cause interference in the DCF receiver. They should preferably be placed apart by at least one metre. At unfavourable locations the reception can leave something to be desired. In these cases it is recommended to place the DCF receiver as high as possible (e.g. in the attic). And finally, don't forget to connect the pull-up resistor to the receiver, shown in figure 3.

(020410-1)

Note:

The software for this project is free and may be obtained via our website at www.elektor-electronics.co.uk, file number 020410-11. Select Free Downloads, then month of publication.